A System-Level Architecture Model for Rapid Prototyping of Heterogeneous Multicore Embedded Systems

Maxime Pelcat, Jean François Nezan, Jonathan Piat, Jerome Croizer, Slaheddine Aridhi

A System-Level Architecture Model for Rapid Prototyping of Heterogeneous Multicore Embedded Systems

Maxime Pelcat, Jean-François Nezan,
Jonathan Piat and Jérôme Croizer
IETR/INSA Rennes/CNRS UMR 6164
{mpelcat, jnezan, jpiat, jcroizer}@insa-rennes.fr

Slaheddine Aridhi
Texas Instruments, CIV Division
saridhi@ti.com

Abstract

Modern embedded systems tend to consist of multiple processors like multicore DSP (Digital Signal Processor) or MPSoC (Multiprocessor System-on-Chip). Static task scheduling for rapid prototyping of parallel embedded systems is different from the dynamic monocore scheduling problem. The communication between cores has a very high impact on the scheduling and the resulting use of hardware resources. Taking into account communication costs and competition can increase excessively the time spent in the prototyping. The System-Level Architecture Model proposed in this paper aims to provide simple system-level descriptions of architectures. The model is expressive enough to enable simulation of modern architecture behaviors. It also reduces the complexity of static mapping-scheduling by precalculating routes between elements.

1 Introduction

Modern multicore embedded systems for mobile phones must execute many complex algorithms (GPS algorithms, advanced telecommunication physical and MAC layers, audio and video codecs...) and still maintain their energy consumption below a few Watts. Most embedded systems alike now have very demanding power consumption constraints. These constraints prevent the hardware designers from increasing the clock speed of their processors. In order to gain computational capacity, most of the modern Systems on Chip (SoC) delegate some algorithms to hardware coprocessors or specialized and power-thrifty cores, leading to complex heterogeneous architectures.

Developing parallel deployments is not straightforward. The Algorithm-Architecture-Matching (AAM) methodology consists in exploring at compile-time both the algorithm and the architecture of a deployment and optimize it for speed, memory, etc... The goal is to ease the system designer deployment choices. The rapid prototyping of a deployment is relevant if simple yet expressive models of both the architecture and the algorithm feed the process. Thierry Granpierre in [1] and Pengcheng Mu in [2] define architecture models which are close to the hardware design. They accurately model data exchanges between processing elements but lead to complex interconnections and involve expensive evaluations of data competition in the exploration process of the deployment. This paper describes an architecture model that matches the behavior of modern architectures but keeps a high level of abstraction. The model will be refered to as the System-Level Architecture Model (S-LAM) in the rest of the paper. S-LAM offers a simple description of modern architectures at system-level and decreases the multicore simulation complexity.

S-LAM defines special communication components to express the degree of accuracy needed in the simulation of a deployment. We focus on the critical aspects of modern multicore systems, which are the computation speed of their processing elements and the data transfer cost of their intercore transfers. S-LAM is developed as part of a rapid prototyping tool named PREESM for Parallel and Real-time Embedded Executives Scheduling Method [3] [4]. PREESM implements the AAM methodology. The S-LAM model does not directly feed the rapid prototyping process. The routes between processing elements are pre-calculated from the S-LAM description to speed up the mapping/scheduling process.

Firstly, Section 2 describes S-LAM and its assets and Section 3 presents the route model. Section 4 then details the algorithm transforming S-LAM into the route model. Finally, Section 5 shows how the route model is simulated and Section 6 gives some experimental results.

2 The System-Level Architecture Model

2.1 S-LAM in the AAM process

the AAM methodology explains how to generate, from graph descriptions of an algorithm and an architecture, a system-level simulation of a deployment and a framework code calling manually-written functions [4]. In the PREESM rapid prototyping tool, a third input named scenario is added. It makes any algorithm usable in combination with any architecture. The scenario contains all the information linking an algorithm and an architecture. It holds timing informations of the execution of any algorithm element (actor) on each architecture component, mapping constraints defining which architecture component can execute each actor, as well as settings for simulation and code generation.

The rapid prototyping of a deployment in PREESM is done by applying a workflow to the inputs. Figure 1 describes a classic workflow which can be applied in the PREESM tool. The dataflow model chosen to describe applications in PREESM is a hierarchical timed graph model [5] derived from the SDF model (Synchronous Data Flow [6]). SDF has the advantage of allowing formal verification of static schedulability. A vertex in SDF is called an actor. The typical number of actors to schedule in PREESM is between a hundred and a thousand. The typical size of an architecture is between a few cores and a few dozens of cores. The purpose of the graph transformations module is to modify the SDF graph generating clusters [7]. The use of clusters aims to reduce the number of actors to schedule in the mapping process and to provide a loop-compressed representation in the generated code. Subsequently, the SDF graph is converted into a Directed Acyclic Graph (DAG), which has a lower expressivity than SDF graph but is a suitable input for the mapping/scheduling [8]. The PREESM mapping/scheduling process [4] generates a deployment by statically choosing a core to execute each actor (mapping) and giving a total order to the actors (scheduling).

As a result of the deployment, code is generated and a Gantt chart of the execution is displayed. The generated code consists of a function call per actor, the static schedule of the actors on each processor, data transfers and synchronizations between processors. The deployment can also feed a SystemC-based [9] simulator for accurate deployment simulations. The functions themselves are hand-written. The three kinds of graphs: algorithm, architecture and workflow are edited by the generic graph editor Graphiti [10].

The simplicity of the architecture description is a key argument to justify the use of a system-level exploration tool like PREESM over a manual approach. Keeping a high expressiveness is also important because most embedded architectures are now heterogeneous, i.e. they contain several types of cores and/or inter-core media. These observations led to the System-Level Architecture Model described in the next sections.

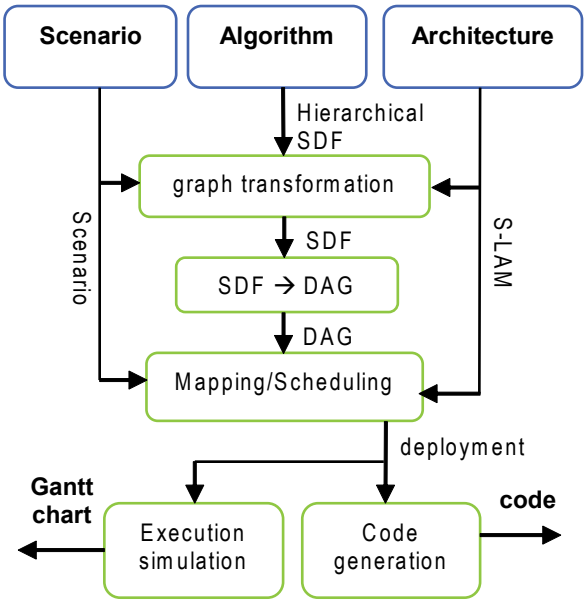

Figure 1. A PREESM rapid prototyping workflow

2.2 The S-LAM operators

An S-LAM description is a topology graph defining the data exchanges between the cores of a heterogeneous architecture. Instead of core, we reuse in this paper the term operator defined in [1]. The reason is that, there is no difference between a core and a coprocessor or an Intellectual Property block (IP) at system-level. They all take input data, process it and return an output data after a given time. All the processing elements are named operators in S-LAM and no more information than their name and type are provided.

The times must be expressed in a single time unit for the whole system even if several clock domains appear. The clock rate does not express anymore the time needed to execute a given actor because of the high complexity of modern CPUs with pipelines, Very Long Instruction Word (VLIW), Single Instruction Multiple Data (SIMD), out of order execution, cache accesses and sometimes Just-in-time (JIT) compilation. The result is that, times in S-LAM are given for each couple $(actor, operatortype)$ in the scenario instead of the the clock rate of each core. IPs, coprocessors and dedicated cores have the ability to execute a given set of actors specified in the scenario as constraints. An operator englobes a local memory to store its processed data.

Transfering data from one operator to another operator via an interconnection actually means transfering the data from one local memory to the other. The next sections explains how to interconnect operators.

2.3 Connecting operators in S-LAM

Two operators can not be merely connected by an edge. Nodes must be inserted in order to provide a data rate to the interconnections. The vertex and edge types in the System-Level Architecture Model, shown in Figure 2, are:

- the parallel node: it models a crossbar with a finite data rate but a perfect capacity to transfer data in parallel,
- the contention node: it models a bus with a finite data rate and contention awareness,
- the RAM: it models a Random Access Memory. It must be connected to a communication node. Operators read and write data through this connection,
- the DMA: it models a Direct Memory Access. A core can delegate the control of communications to a DMA via a set-up link. DMA must also be connected to a communication node,
- the directed (resp. undirected) link: it shows that data can be transfered between two components in one (resp. both) direction(s),
- the set-up link: only between an operator and a DMA or a RAM. It shows that an operator can access a DMA or RAM resource and gives the time needed for setting-up.

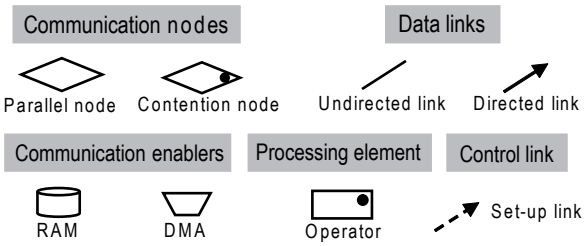

Figure 2. The elements of S-LAM

In the rest of the paper, Parallel and contention nodes are both called communication nodes. Directed and undirected links are both called data links. A black dot in a component in Figure 2 shows that contention is taken into account. The contention expresses a sequential behaviour. An operator has contention and it can not execute more than one actor at a time. A contention node has contention and it can not transfer more than one data at a time. The elements with contention are the ones taking time during the deployment simulation because a scheduling of their actors or transfers must be computed. Connecting an operator to a DMA via a set-up link means that the operator delegates to the DMA the controls of transfers going through communication nodes connected to the same DMA via a data link.

S-LAM consists in components and interconnections, each with a type and a few properties. This model naturally fits in the IP-XACT model, an IEEE standard from the SPIRIT consortium [11] intended to store XML descriptions of any type of architecture.

2.4 Examples of S-LAM descriptions

Figure 3 shows the simplified block diagram of a Texas Instruments TMS320TCI6488 processor [12]. This tri-core processor has (among others), co-processors named TCP2 and VCP2 respectively accelerating the turbo decoding and Viterbi decoding of a bitstream. The processor offers a DDR2 interface to connect a Double Data Rate RAM and a standardized serial RapidIO interface to connect another device.

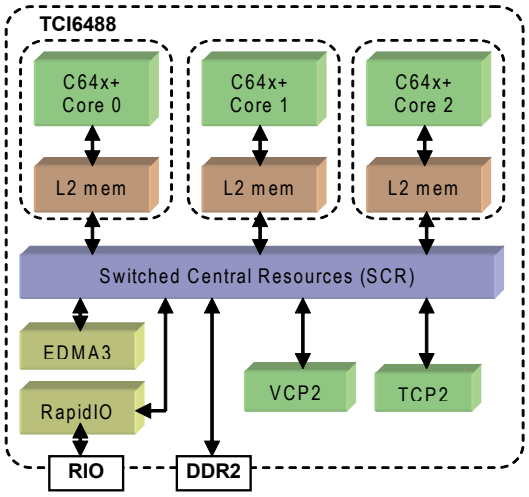

Figure 3. Block diagram of a TMS320TCI6488

Figure 4 shows the S-LAM of two TMS320TCI6488 at 1GHz connected via their RapidIO links. In this example, the choice is made not to consider contention on the SCR; this choice can be changed with a few clicks in PREESM. The RapidIO link at 1Gbit/s = 0.125GByte/s = 0.125 Byte/-cycle is the bottleneck of the architecture. It is represented by a single contention node. Each GEM operator contains both a C64x+ core and a local L2 memory. They delegate both their intra and inter-processor transfers to a DMA. The local transfers were benchmarked in [13]. The results were

a data rate of 2GByte/s and a set-up time of 2700 cycles, values put here in S-LAM.

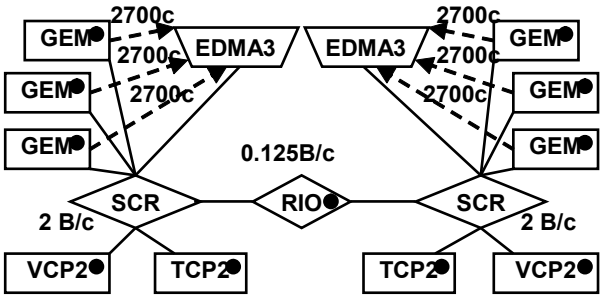

Figure 4. S-LAM description of a board with 2 TMS320TCI6488

As an example of S-LAM expressiveness, Figure 5 shows a unidirectional ring architecture where each operator can communicate only with one of its neighbors. This kind of architecture will be used to test the mapping process in the last section.

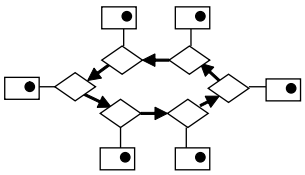

Figure 5. S-LAM description of a ring architecture

The S-LAM gives a simplified description of an architecture focusing on the real bottlenecks of the design. In order to fasten the rapid prototyping, the S-LAM model does not directly feed the PREESM mapper/scheduler. It is first transformed into a so-called route model.

3 The route model

3.1 The route model in the AAM process

The PREESM rapid prototyping tool maps and schedules statically an algorithm onto an architecture. The PREESM mapper/scheduler evaluates many deployments while making its mapping and scheduling choices. This problem is NP complete [14] and must be solved by heuristics [4]. The heuristic part makes the mapping choices and generates deployments while the Architecture Benchmark Computing (ABC) part simulates the deployments. This structure was presented in [4] and is extended in this paper to handle S-LAM architectures (Figure 6).

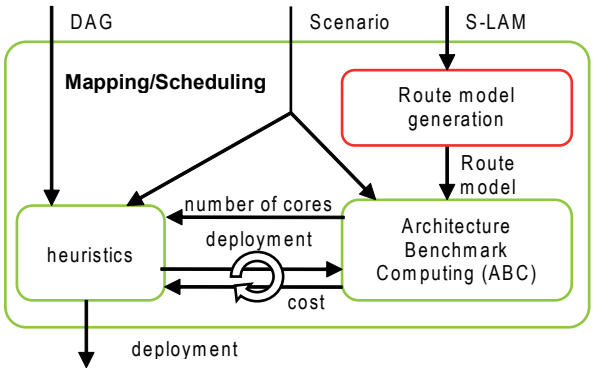

Figure 6. The route model in the AAM process

Simulating the data transfers during mapping and scheduling makes the simulation of the deployments quite complex because vertices representing transfers are dynamically added to and deleted from the graph. Adding new cores to the architecture should not increase the mapping and scheduling complexity exponentially. To achieve this goal, the S-LAM is transformed into the route model before mapping and scheduling as illustrated figure 6.

3.2 The inter-operator routes

A route step represents an interconnection between two operators. A route is a list of route steps and represents a way to transfer data between two operators, directly connected or not. A direct route between two operators is a route containing a single route step. It means that the data is transfered directly from one operator to another without being copied by an intermediate operator. In a totally connected architecture, there is always a direct route between two given operators. In a more general case, we need to build routes to handle the multiple transfers of the same data along a route in the generated code.

Given the S-LAM model of Section 2, there are three types of route steps interconnecting operators. The types, shown in Figure 7 are:

- The message passing route step: the source operator sends data to the target operator via message passing. The data flows through one or several route steps before reaching the target,
- The DMA route step: the source operator delegates the transfer to a DMA that sends data to the target operator via message passing,
- The shared memory route step: the source writes data into a shared memory and the target then reads the

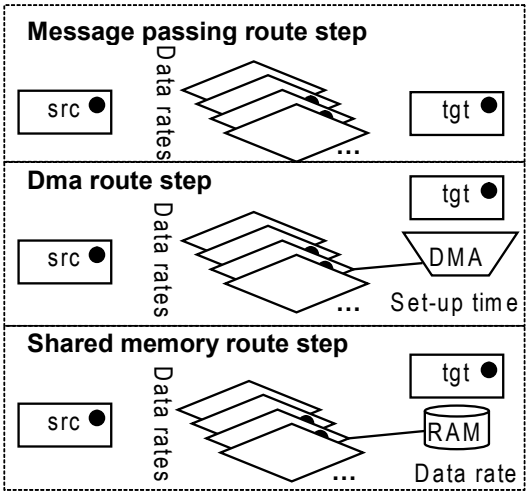

Figure 7. The types of route steps

data. The position of the RAM element (the communication node to which it is connected) is important because it selects the communication nodes used while writing and the ones used while reading.

Routes can be created from these routes steps to interconnect operators. The route model contains two parts: an operator set containing all the operators in the architecture and a map named routing table assigning the best route to each couple of operators $(source, target)$. The route model speeds up the deployment simulations because it gives immediately the best route between two operators without looking through the S-LAM graph. The transformation of S-LAM into route model is explained in next section.

4 Transforming S-LAM into the route model

S-LAM was developed in order to simplify the multi-core scheduling problem of the PREESM tool. We will now study the route precalculation that implies this complexity reduction.

4.1 Overview of the transformation

The route model generation from figure 6 is detailed in figure 8. Transforming S-LAM into routes is done in three steps: route steps are generated first, followed by the generation of direct routes and finally by the composed routes. Each step is detailed in the following.

4.2 Generating a route step

From a source operator, a target operator and a list of communication nodes connecting these operators, we can

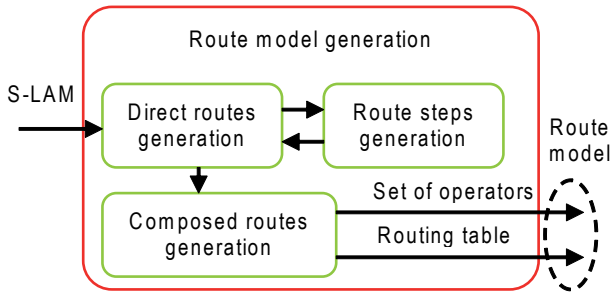

Figure 8. The route model generation

generate a route step with one of the types defined in Section 3.2. While creating the route step, the communication nodes are scanned and connections to DMA or RAM vertices are searched to determine the type of the current route step. If a DMA is found, its incoming set-up links are searched and if none of them have the same source as the current route step, the DMA is not taken into account. Transfers not driven by DMA are thus allowed and they can share a communication node with DMA-driven transfers. Contentions between all transfers on a given communication node can be simulated.

4.3 Generating direct routes from the graph model

Using the route step generation, the direct route generation looks through the graph starting from each operator src. The algorithm displayed in Figures 9 and 10 scans the communication nodes and keeps lists of the previously visited nodes. When an operator tgt is met, it generates a route step using the method in Section 4.2. If the new route step has a lower cost than the one (if any) in the table, a new route containing only the new step is put into the table. The cost of a route is defined as the sum of the costs of its route steps. The cost of a route step depends on the route step type and is calculated using a typical data size set in the scenario.

```
1)foreach operator src in operators
2) foreach interconnection i in
    outgoing and undirected edges of src
3)  if the edge other end is a node n
4)   add the node n to a list l of
      already visited nodes;
5)   call exploreroute(src,n,l);
6)  endif;
7) endforeach;
8)endforeach;
```

Figure 9. Direct routes generation

```
/* this recursive function scans the
communication nodes and adds a route
when reaching a target operator*/
9)function exploreroute
   (operator src,node n, nodelist l)
10)foreach interconnection i in
    outgoing and undirected edges of n
11) if the edge other end is a node n2
12)  create a new list l2 containing
      all the elements of l;
13)  add n2 to l2;
14)  call exploreroute(src,n2,l2);
15) elseif the other end of the edge is
     an operator tgt
16)  generate a route step from src, tgt and
      the list of nodes l;
17)  get the table current best route
      between src and tgt;
18)  if the new route step has a lower cost
      than the table route step;
19)   set it as the table best route
       from src to tgt;
20)  endif;
21) endif;
22)endforeach;
23)endfunction;
```

Figure 10. Exploring a direct route

The complexity of the algorithm is $O(NC^2)$ where N is the number of operators in the graph and C the number of communication nodes. After the direct route generation, the routing table contains all the direct routes between interconnected operators.

4.4 Generating the complete routing table

In our model, the operators are not necessarily totally interconnected. The routes with multiple route steps are then built using a Floyd-Warshall algorithm [15] provided Figure 11. The route between a source src and a target tgt is computed by composing previously existing routes and keeping the one with the lowest cost.

The Floyd-Warshall algorithm keeps the best route between two given operators with a complexity of $O(N^3)$ and is proved to be optimal for such a routing table construction. The complexity of the routing table computation is not really problematic in our case because the architectures are always a few cores and the routing table construction of a reasonably interconnected architecture with 20 cores was benchmarked at under 1s compared to mapping/scheduling requiring several minutes (see Section 6). The route model simply consists in this table and the set of operators in the input S-LAM model. If the table is not completed, i.e. a couple of operators (src, tgt) exists which best route does not exist in the routing table, then the architecture is not totally connected via routes. The PREESM mapper and scheduler does not handle such architectures. Thanks to the S-LAM model, PREESM can stop and return an error before starting the mapping and scheduling process. The definition of routes allows us to study transfers simulation in PREESM.

```
1)foreach operator k in operators
2) foreach operator src in operators
3)  foreach operator tgt in operators
4)   get the table best route from src to k;
5)   get the table best route from k to tgt;
6)   compose the 2 routes in a new route
      from src to tgt;
7)   evaluate the composition;
8)   compare the table best route
      from src to tgt with the composition;
9)   if the composition has a lower cost
10)    set it as the table best route
       from src to tgt;
     endif;
11) endforeach;
12)endforeach;
13)endforeach;
```

Figure 11. Floyd-Warshall algorithm: computing the routing table

5 Simulating a deployment using the route model

As shown in Figure 6, the PREESM mapper/scheduler is divided between mapping heuristics and deployment evaluation [4]. The most common scheduling case is to evaluate and minimize the deployment latency. A Gantt chart of the deployment execution is then constructed. Depending on the route step type, different transfer simulations are inserted into the Gantt chart in addition to actor simulations. The next sections describe the simulation of a tranfer of a number of bytes for each type of route step.

5.1 The message passing route step simulation with contention nodes

Part 1 of Figure 12 shows the simulation of a single message passing transfer between actor1 mapped on src and actor2 mapped on tgt. The transfer blocks the two contention nodes in the route step during the transfer time. The data rate of the transfer (in Byte per cycle) is the lowest data rate

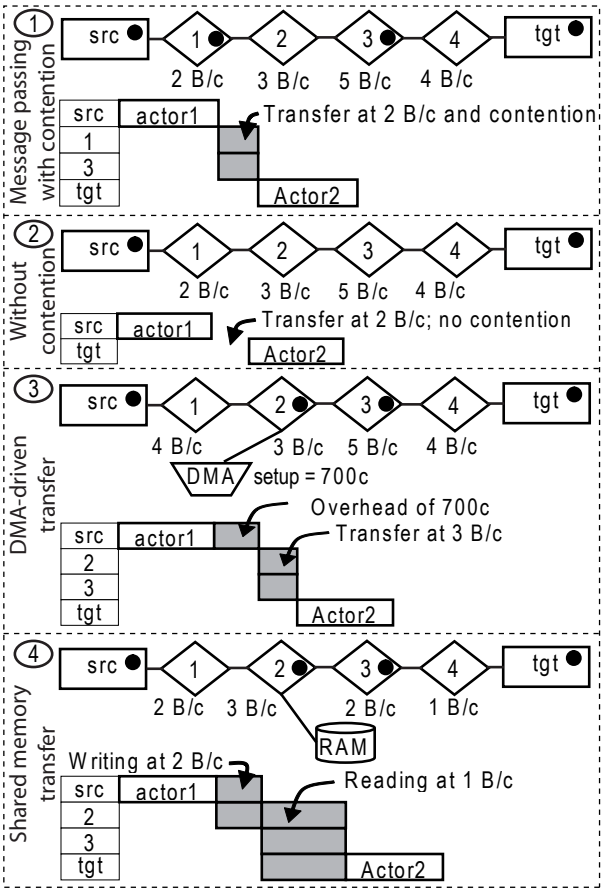

Figure 12. Impact of route types in the simulation of a transfer

of the communication nodes (which is the bottleneck of the communication).

5.2 The message passing route step simulation without contention nodes

If no contention node is present like in Part 2 of Figure 12, there is no line in the Gantt chart for the transfer but the actor2 is delayed until the transfer is complete. This model is equivalent to the ones used by Yu-Kwong Kwok in his mapping and scheduling algorithms [16]. Consequently, parallel nodes take into account the transfer delays but just ignore the contentions.

5.3 The DMA route step simulation

The simulation of a DMA transfer is shown in Part 3 of Figure 12. The set-up transfer overhead is mapped onto the source operator and the transfer is then equivalent to a message passing. In practice the overhead corresponds to the set-up time of the DMA, i.e. the time to write the transfer description into the DMA registers.

5.4 The shared memory route step simulation

The simulation of a shared memory transfer is shown in Part 4 of Figure 12. First, the source operator writes the data to the shared memory and then the target operator reads it. The left-hand side communication nodes are occupied during the writing and the right-hand side ones during the reading. The writing and reading data rates can be different. They are limited by the maximum data rates of the memory and of the communication nodes.

6 Experimental results

The simulation accuracy of the transfers is tested by generating code with PREESM for a 3GPP Long Term Evolution (LTE [17]) algorithm called Random Access CHannel Preamble Detection (RACH-PD) [13]. The target architecture is the previously presented TMS320TCI6488. This algorithm has 240 vertices and 457 edges. The RACH-PD algorithm is parallelized on two cores of the TMS320TCI6488 leading to 68 synchronized EDMA3 transfers. The correctness of the algorithm results is tested. The PREESM latency estimation is 6.92 Mcycles and the achieved latency is 6.75 Mcycles. The error of 2.5% results from the sum of errors on task execution times, on transfers times, on synchronizations and on the duration of some automatically generated memory copies. The rapid prototyping process gives a good approximation of the algorithm execution.

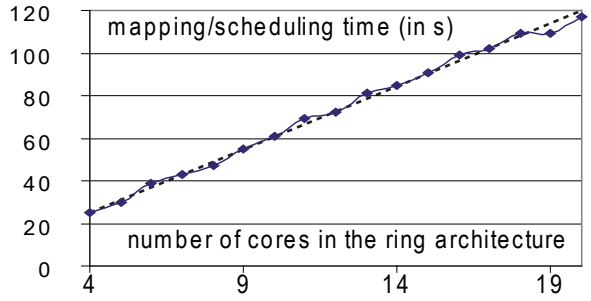

Figure 13. mapping/scheduling time of an LTE algorithm on ring architectures with increasing number of cores

Deployments are simulated under PREESM to demonstrate the suitability of the routing process. The algorithm is the 3GPP LTE uplink physical layer reception in a base station. Our description of this algorithm is highly parallel

and contains 398 actors and 542 transfers. The architectures are unidirectional rings like in Figure 5 with increasing number of cores between 4 and 20. The result of this test is shown in Figure 13. We see that the time needed to map and schedule the algorithm increases linearly with the number of cores N thanks to the pre-calculated routes. Such a result in $O(N)$ could not be obtained if an increasing architecture graph would be scanned for each transfer evaluation. As evoked in Section 4.4, the routing time remains negligible compared to the mapping and scheduling process for the target number of cores. This result shows the suitability of the pre-calculated route model.

7 Conclusion and perspectives

We described here a simple yet accurate System-Level Architecture Model. It is designed to keep a high expressiveness and to focus on the data transfer bottlenecks of the architecture. This architecture model can be efficiently transformed into a route model before feeding a rapid prototyping process. The route model speeds up the process of mapping and scheduling an algorithm onto an architecture.

PREESM makes the static mapping and scheduling choices and generates a deployment. This deployment can then be sent to a SystemC-based simulation tool. The tool computes cycle accurate simulations of code executions and completes the rapid prototyping process. Using PREESM and SystemC together combines the high-level S-LAM architecture model for mapping and scheduling heuristics and the cycle accurate SystemC model for the evaluation of the chosen deployment. The combination of both models provides a precise and fast rapid prototyping process.

References

[1] Thierry Grandpierre, *Modélisation d'architectures paralleles hétérogenes pour la génération automatique dexécutifs distribués temps réel optimisés*, Ph.D. thesis, INRIA, 2000.

[2] Pengcheng Mu, *Rapid Prototyping Methodology for Parallel Embedded Systems*, Ph.D. thesis, INSA Rennes, 2009.

[3] "PREESM : Available Online," http://sourceforge.net/projects/preesm/.

[4] Maxime Pelcat, Pierrick Menuet, Slaheddine Aridhi, and Jean-Francois Nezan, "Scalable Compile-Time scheduler for multi-core architectures," in *DATE*, 2009.

[5] Jonathan Piat, Shuvra S. Bhattacharyya, Maxime Pelcat, and Mickael Raulet, "Multi-core code generation from interface based hierarchy," in *DASIP (to appear)*, 2009.

[6] E.A. Lee and D.G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[7] Jos Luis Pino and Edward A Lee, "Hierarchical static scheduling of dataflow graphs onto multiple processors," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 2643—2646, 1995.

[8] Jos Luis Pino, Shuvra S Bhattacharyya, and Edward A Lee, "A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs," *Laboratory, University of California at Berkeley*, pp. 95—36, 1995.

[9] "Open SystemC initiative web site," http://www.systemc.org/home/.

[10] "Graphiti Editor : Available Online," http://sourceforge.net/projects/graphiti-editor/.

[11] "The SPIRIT consortium, IP-XACT v1.4: A specification for XML meta-data and tool interfaces," Mar. 2008.

[12] "TMS320TCI6488 DSP platform, texas instrument product bulletin (sprt415)," 2007.

[13] Maxime Pelcat, Slaheddine Aridhi, and Jean Francois Nezan, "Optimization of automatically generated multi-core code for the LTE RACH-PD algorithm," *0811.0582*, Nov. 2008, DASIP 2008, Bruxelles : Belgique (2008).

[14] Michael R. Garey and David S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., 1990.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, The MIT Press, 2nd edition, Sept. 2001.

[16] Yu-Kwong Kwok, *High-performance algorithms of compile-time scheduling of parallel processors*, Ph.D. thesis, Hong Kong University of Science and Technology, 1997.

[17] "3GPP technical specification group radio access network; evolved universal terrestrial radio access (E-UTRA) (Release 8), 3GPP, TS36.211 (V 8.1.0)," 2008.